**SAD Lab**

**EXPERIMENT NO. 2**

**Aim**: To learn Case study for SDLC.

**Theory**:

1. What is a secure SDLC and why is it important?

**Secure SDLC**

The software development life cycle (SDLC) framework maps the entire development process. It includes all stages such as planning, design, build, release, maintenance, and updates, as well as the replacement and retirement of the application when the need arises.

Security applies at every phase of the software development life cycle (SDLC) and needs to be at the forefront of your developers’ minds as they implement your software’s requirements. With dedicated effort, security issues can be addressed in the SDLC pipeline well before deployment to production. This reduces the risk of finding security vulnerabilities in your app and works to minimize the impact when they are found.

The secure SDLC (SSDLC) builds on this process by incorporating security in all stages of the life cycle. Teams often implement an SSDLC when transitioning to DevSecOps. The process involves applying security best practices alongside functional aspects of development, and securing the development environment.

Secure SDLC’s aim is not to completely eliminate traditional security checks, such as penetration tests, but rather to include security in the scope of developer responsibilities and empower them to build secure applications from the outset.

**Importance of secure SDLC**

It is a common belief that security requirements and testing inhibit the development process. However, a secure SDLC provides an effective method for breaking down security into stages during the development process. It unites stakeholders from development and security teams with a shared investment in the project, which helps to ensure that the software application is protected without being delayed.

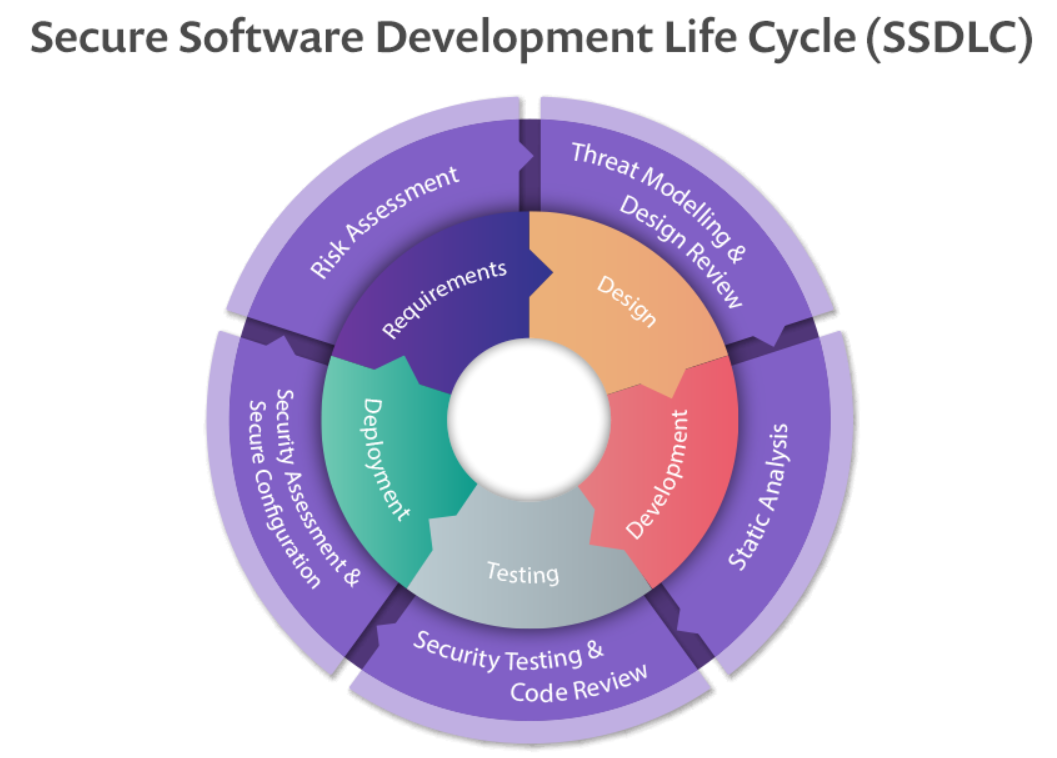
Developers may start by learning about the best secure coding frameworks and practices. They should also look into using automated tools to identify security risks within the code they write and to detect security vulnerabilities in the open source libraries they bring into their projects.

In addition, the management team may use a secure SDLC as a vehicle to implement a strategic methodology to create a secure product. For example, managers can perform a gap analysis to gain insight into which security activities or policies currently exist, which are absent, and to see how effective they are at each stage of the SDLC.

To achieve a streamlined SSDLC and ensure software shipping deadlines are not missed, it is necessary to establish and enforce security policies that help address high-level issues like compliance without requiring manual review or manual intervention. To achieve this, some organizations choose to hire security experts to evaluate security requirements and to create a plan that will help the organization improve its security preparedness.

1. What are the phases involved in secure SDLC?

Implementing SDLC security affects every phase of the software development process. It requires a mindset that is focused on secure delivery, raising issues in the requirements and development phases as they are discovered. This is far more efficient and much cheaper than waiting for these security issues to manifest in the deployed application. Secure software development life cycle processes incorporate security as a component of every phase of the SDLC.



Each phase of the SDLC must contribute to the security of the overall application. This is done in different ways for each phase of the SDLC, with one critical note: Software development life cycle security needs to be at the forefront of the entire team’s minds.

**PHASE 1: REQUIREMENTS**

In this early phase, requirements for new features are collected from various stakeholders. It’s important to identify any security considerations for functional requirements being gathered for the new release.

* Sample functional requirement: Users need the ability to verify their contact information before they are able to renew their membership.
* Sample security consideration: Users should be able to see only their own contact information and no one else’s.

**PHASE 2: DESIGN**

This phase translates in-scope requirements into a plan of what this should look like in the actual application. Here, functional requirements typically describe what should happen, while security requirements usually focus on what shouldn’t.

* Sample functional design: Page should retrieve the user’s name, email, phone, and address from CUSTOMER\_INFO table in the database and display it on screen.
* Sample security concern: We must verify that the user has a valid session token before retrieving information from the database. If absent, the user should be redirected to the login page.

**PHASE 3: DEVELOPMENT**

When it’s time to actually implement the design and make it a reality, concerns usually shift to making sure the code is well-written from the security perspective. There are usually established secure coding guidelines as well as code reviews that double-check that these guidelines have been followed correctly. These code reviews can be either manual or automated using technologies such as static application security testing (SAST).

Secure coding guidelines, in this case, may include:

* Using parameterized, read-only SQL queries to read data from the database and minimize chances that anyone can ever commandeer these queries for nefarious purposes
* Validating user inputs before processing data contained in them
* Sanitizing any data that’s being sent back out to the user from the database
* Checking open source libraries for vulnerabilities before using them

**PHASE 4: VERIFICATION**

The verification phase is where applications go through a thorough testing cycle to ensure they meet the original design & requirements. This is also a great place to introduce automated security testing using a variety of technologies. The application is not deployed unless these tests pass. This phase often includes automated tools like CI/CD pipelines to control verification and release.

Verification at this phase may include:

* Automated tests that express the critical paths of your application
* Automated execution of application unit tests that verify the correctness of the underlying application
* Automated deployment tools that dynamically swap in application secrets to be used in a production environment

**PHASE 5: MAINTENANCE AND EVOLUTION**

Vulnerabilities that slipped through the cracks may be found in the application long after it’s been released. These vulnerabilities may be in the code developers wrote, but are increasingly found in the underlying open-source components that comprise an application. This leads to an increase in the number of “zero-days”—previously unknown vulnerabilities that are discovered in production by the application’s maintainers.

These vulnerabilities then need to be patched by the development team, a process that may in some cases require significant rewrites of application functionality. Vulnerabilities at this stage may also come from other sources, such as external penetration tests conducted by ethical hackers or submissions from the public through what’s known as “bug bounty” programs. Addressing these types of production issues must be planned for and accommodated in future releases.

**Observation:**

*A Case Study of the Application of the Systems Development Life Cycle (SDLC) in 21st Century Health Care: Something Old, Something New?*

**CASE STUDY SETTING**

The study examines the selection of a software package by a medium-size regional hospital for use in the Home Health segment of their organization. The 149-bed facility is a state-of-the-art institution, as 91% of their 23 quality measures are better than the national average. Services offered include Emergency Department, Hospice, Intensive Care Unit (ICU), Obstetrics, Open Heart Surgery, and Pediatrics.

**INTRODUCTION**

The systems development life cycle (SDLC) is one of the oldest and most widely used software development and acquisition methods in the IT arena. This paper describes the use of the SDLC in a real-world healthcare setting involving a principal component of a regional hospital care facility. The paper can be used as a pedagogical tool in a systems analysis and design course, or in an upper-division or graduate course as a case study.

**BACKGROUND**

The SDLC has been a part of the IT community since the inception of the digital computer. The Waterfall Model (Boehm, 1976) is one of the most well-known forms. At its heart it remains a robust methodology for developing software and systems. As this paper will show, following the steps and stages of the methodology is still a valid method of ensuring the successful deployment of software.

McMurtrey (1997) came up with a generic SDLC involving the phases of Analysis, Design, Coding, Testing, Implementation, and Maintenance. Whitten and Bentley (2008) text, in its present form, still breaks up the process into eight stages. The naming conventions used in the newer text are almost synonymous with those in the older work. Although not immune to criticism, Hoffer, George, and Valacich believe that the view of systems analysis and design taking place in a cycle continues to be pervasive and true.

**ANALYSIS**

The Home Health portion of General Hospital had been reorganized as a subsidiary unit located near the main hospital in its own standalone facility. The software they were using was at least seven years old and could simply not keep up with all the changes in billing practices and Medicare requirements and payments. Our purpose in helping General was twofold to modernize their operations with current technology and to provide the best patient care.

In the Requirements Analysis stage of a potential acquisition, great care is taken to ensure that the proposed system meets the objectives put forth by management. MEDITECH (Medical Information Technology, Inc.) has been a leading software vendor in the healthcare informatics industry for 40 years. All Meditech platforms are certified EMR/EHR systems. This was the first, and one of the most important requirements, at least from a technological viewpoint. Point of Care Documentation (POC) documentation in patients' rooms is a recent shift in technology use in hospitals.

POC documentation reduces inefficiencies, decreases the probability of errors, promotes information transfer, and encourages the caregiver to be at bedside or, in the case of home care, on the receiving end of the transmission. OASIS Analyzer allows staff to work more intelligently, allowing them to easily analyze outcomes data in an effort to move toward improved clinical and financial results.

The chosen software package must have an entryway for the attending, resident, or primary caregiver physician to interact with the system. Such a gateway will facilitate efficient patient care by enabling the physician to have immediate access to critical patient data and history. Vendor A had an Implementation/Installation Team to assist with that stage of the software deployment. Vendor C sponsored an annual User Conference where users could share experiences with using the product. Vendor E's offering was part of an enterprise solution and could be synchronized with a PDA or smartphone.

**FINDINGS**

In this case study of software selection, the researchers did not have to proceed through each step of the SDLC since the software products already existed. In a similar vein, the coding, testing, and debugging of program modules had too been performed by each vendor candidate. After painstakingly analyzing all the wares, features, pros and cons, and costs and benefits associated with each product, we were now ready to make a choice.

As is taught in SAD classes, the implementation stage of the SDLC usually follows one of four main forms. Direct Installation, where the old system is simply removed and replaced with the new software, perhaps over the weekend. Parallel Installation, when the old and new systems are run side-by-side until at some point use of the former software is eliminated. Single Location Installation (or the Pilot approach) involves using one site (or several sites if the software rollout is to be nationwide or international) as beta or test installations.

**CONCLUSION**

Two business school professors, one an MIS scholar and the other retired from the accounting faculty, were called upon by a local hospital to assist with the procurement of software for the Home Health area. These academics were up to the challenge, and pleasantly assisted the hospital in their quest. While both researchers hold terminal degrees, each learned quite a bit from the application of principles taught in the classroom (e.g., the SDLC) to the complexities surrounding real-world utilization of them. Great insights were gained, in a variety of areas, and have since been shown as relevant to future practitioners (i.e., students) in the business world. It is hoped that others, in both academe and commerce, will benefit from the results and salient observations from this study.

**Difference between software development life cycle and secure software development life cycle**

SDLC is defined as a life cycle for software development. SDLC is a project development phase that is pursued by a software business. It provides a comprehensive plan detailing how unique software will be designed, managed, replaced and improved. The life cycle describes a methodology for changes in software quality and the overall production process.The main objective of SDLC is to provide high-quality software as a part of the clients requirements. SDLC contains six phases which are: Requirement gathering, Designing, Coding, Testing, and Maintenance. Thus, it's vital to follow these phases to develop the Product in a systematic manner. For instance when a program has to be developed, developers can split the features of the product and develop what they want. If Some of the teams start to design the program, others decide first to write the coding part or the documentation part, this can result in a failure on the developers and also not to develop the required project.

Even though SSDLC has the same phases of software development life cycle, it also includes security at every phase. The use of secure coding methods also requires this approach. Security is not only a goal, however a core concept that is incorporated at each progression of the product's strategy and design. It is crucial to provide security in the SDLC when it recalls minor and critical product highlights. It is solved by coordinated activities between programmers, project managers, and security engineers on a non-stop and continuous basis. The goal of Secure SDLC is not to remove conventional security checks, such as penetration tests, entirely, but rather to provide security in the framework of developer responsibilities and enable them from the beginning to create secure applications.

**Conclusion**:

Secure SDLC is important because application security is important. The days of releasing a product into the wild and addressing bugs in subsequent patches are gone. Developers now need to be cognisant of potential security concerns at each step of the process. This requires integrating security into your SDLC in ways that were not needed before. As anyone can potentially gain access to your source code, you need to ensure that you are coding with potential vulnerabilities in mind. As such, having a robust and secure SDLC process is critical to ensuring your application is not subject to attacks by hackers and other nefarious users.